Project 2

Daniel Fredin, Junhan Li, & Eric Chen

Problem 1

part(a)

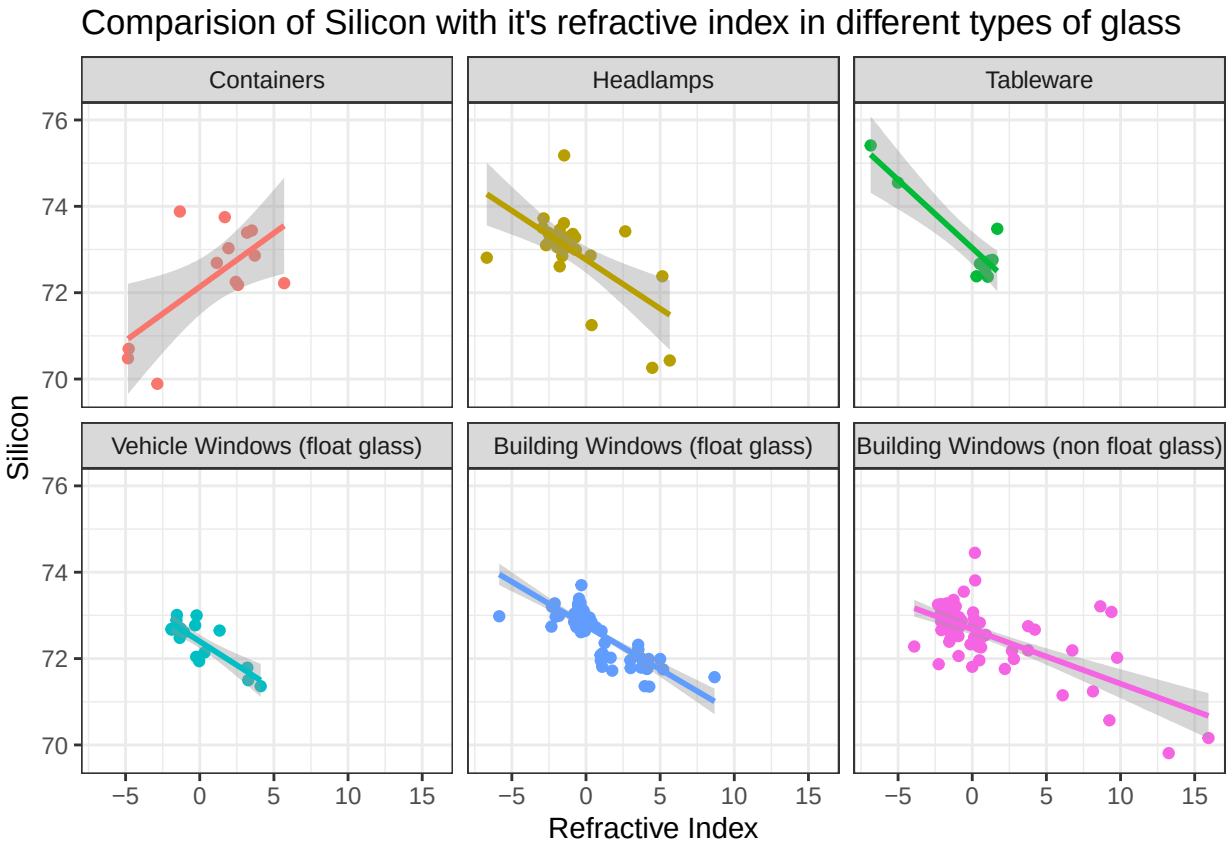
```
# Correlation for all numerical variables
cor(glass[,c("Si", "RI", "Na", "Mg", "Al", "K", "Ca", "Ba", "Fe")])
```

```
##             Si            RI          Na           Mg          Al            K
## Si  1.00000000 -0.5420521997 -0.06980881 -0.165926723 -0.00552372 -0.193330854
## RI -0.54205220  1.0000000000 -0.19188538 -0.122274039 -0.40732603 -0.289832711
## Na -0.06980881 -0.1918853790  1.00000000 -0.273731961  0.15679367 -0.266086504
## Mg -0.16592672 -0.1222740393 -0.27373196  1.000000000 -0.48179851  0.005395667
## Al -0.00552372 -0.4073260341  0.15679367 -0.481798509  1.00000000  0.325958446
## K  -0.19333085 -0.2898327111 -0.26608650  0.005395667  0.32595845  1.000000000
## Ca -0.20873215  0.8104026963 -0.27544249 -0.443750026 -0.25959201 -0.317836155
## Ba -0.10215131 -0.0003860189  0.32660288 -0.492262118  0.47940390 -0.042618059
## Fe -0.09420073  0.1430096093 -0.24134641  0.083059529 -0.07440215 -0.007719049
##            Ca            Ba           Fe
## Si -0.2087322 -0.1021513105 -0.094200731
## RI  0.8104027 -0.0003860189  0.143009609
## Na -0.2754425  0.3266028795 -0.241346411
## Mg -0.4437500 -0.4922621178  0.083059529
## Al -0.2595920  0.4794039017 -0.074402151
## K  -0.3178362 -0.0426180594 -0.007719049
## Ca  1.0000000 -0.1128409671  0.124968219
## Ba -0.1128410  1.0000000000 -0.058691755
## Fe  0.1249682 -0.0586917554  1.000000000
```

```
# Select the highest correlated variables
glass_new <- glass %>%
  select(Si, RI, Ca, type)

# Rename for facet wrap labels
glass_names <- as_labeller(
  c('Con' = "Containers",
    'Head' = "Headlamps",
    'Tabl' = "Tableware",
    'Veh' = "Vehicle Windows (float glass)",
    'WinF' = "Building Windows (float glass)",
    'WinNF' = "Building Windows (non float glass)")
)
```

```
# Scatter plot of Si vs RI for each glass type
ggplot(data= glass_new, aes(x=RI, y= Si, color = type)) +
  geom_point() +
  geom_smooth(method = lm, se = T) +
  facet_wrap(~type, labeller = glass_names) +
  labs(title = "Comparision of Silicon with it's refractive index in different types of glass",
       x = "Refractive Index",
       y = "Silicon",
       color = "Glass Types") +
  theme_bw() +
  theme(legend.position = "none") +
  guides(color = guide_legend(nrow = 1))
```



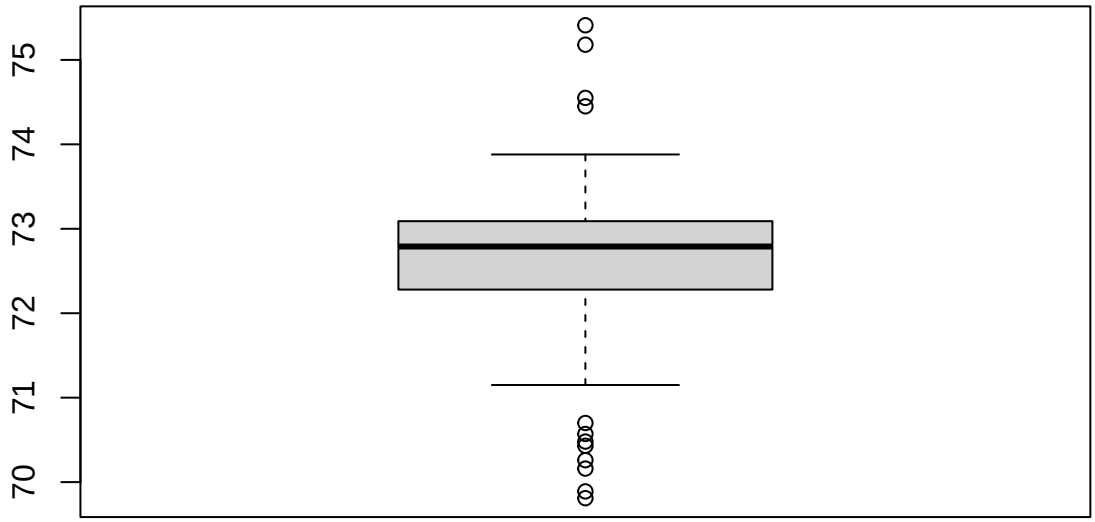

```
# We will first check for the normality of dependent variable
shapiro.test(glass$Si)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  glass$Si
## W = 0.91966, p-value = 2.175e-09
```

We choose the variable RI (refractive index) and Si as our x and y axis to illustrate their correlations.

- 2 Paragraphs
  - why we choose variables
  - what you see from graph

```
boxplot(glass$Si)
```



Part (b)

```
set.seed(123)

# Regular linear regression with all variables
glass_model1 <- lm(Si ~., data = glass)
predictions <- glass_model1 %>% predict(glass)
model1_results <- data.frame( R2 = R2(predictions, glass$Si),
            RMSE = RMSE(predictions, glass$Si),
            MAE = MAE(predictions, glass$Si))
model1_results
```

```
##          R2       RMSE        MAE
## 1 0.9849829 0.09469422 0.06631551
```

```r
# Regular linear regression with 4 variables from our visualization
glass_model2 <- lm(Si  ~ RI + factor(type) + Na + Ca , data = glass)
predictions <- glass_model2 %>% predict(glass)
model2_results <- data.frame( R2 = R2(predictions, glass$Si),
             RMSE = RMSE(predictions, glass$Si),
             MAE = MAE(predictions, glass$Si))
model2_results
```

```
##          R2      RMSE       MAE
## 1 0.5502586 0.5182167 0.3477134
```

```r
# Training and testing method (SEEMS LIKE OUR BEST MODEL BECAUSE OF LOWER RMSE)
training <- glass$Si %>%
    createDataPartition(p = .75, list = FALSE)
train.data  <- glass[training, ]
test.data <- glass[-training, ]

glass_model3 <- lm(Si ~ Na + Mg + Al + K + Ca + Ba + Fe, data = train.data)
predictions <- glass_model3 %>% predict(test.data)
model3_results <- data.frame( R2 = R2(predictions, test.data$Si),
            RMSE = RMSE(predictions, test.data$Si),
            MAE = MAE(predictions, test.data$Si))
model3_results
```

```
##          R2       RMSE        MAE
## 1 0.9930191 0.07494135 0.06087369
```

```r
# varImp(glass_model3)




# # model 4 will use k-cross validation
# ctrl <- trainControl(method = "cv", number = 10)
# glass_model4 <- train(Si ~ ., data = glass, method = "lm", trControl = ctrl)
# print(glass_model4)
#
# # model 5 will use repeated k-cross validation
# ctrl_repeat <- trainControl(method = "repeatedcv", number = 10,  repeats = 3)
# glass_model5 <- train(Si ~ ., data = glass,
#                       method = "lm", trControl = ctrl_repeat)
# print(glass_model5)
```

Argue your choice - Perform optimization in R, compute the regression coefficients

The lower the value for AIC, the better the fit of the model. The absolute value of the AIC value is not important. It can be positive or negative.

Preconlcusion: Model 3 is a lot better. Based on the RMSE values we generated for the three models above, the model 3 where we used 75% training data model is our best model

Part (c)

- (i) Coefficient of determination

```
RMSE <- model3_results[1]
RMSE
```

```
##          R2
## 1 0.9930191
```

The coefficient of determination, also known as R-squared, is a metric used in regression analysis to evaluate how well a model fits the data. It indicates the proportion of the dependent variable's variability that can be explained by the independent variables. R-squared ranges from 0 to 1, where 0 signifies that the independent variables have no influence on the variation, and 1 means they explain all of it. Our best model had an impressive R-squared value of 0.9930, indicating that 99.30% of the dependent variable's variance is captured by the independent variables.

- (ii) Least-squares estimates for the regression line

```
summary(glass_model3)
```

```
##
## Call:
## lm(formula = Si ~ Na + Mg + Al + K + Ca + Ba + Fe, data = train.data)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.73472 -0.04143  0.01324  0.06600  0.20186
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 98.81728    0.35017  282.20  < 2e-16 ***
## Na          -0.94187    0.01604  -58.73  < 2e-16 ***
## Mg          -0.99212    0.01336  -74.28  < 2e-16 ***
## Al          -1.00362    0.02898  -34.64  < 2e-16 ***
## K           -0.98008    0.02047  -47.88  < 2e-16 ***
## Ca          -0.97698    0.01402  -69.69  < 2e-16 ***
## Ba          -0.96023    0.02492  -38.53  < 2e-16 ***
## Fe          -0.34948    0.09497   -3.68 0.000322 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.1123 on 154 degrees of freedom
## Multiple R-squared:  0.9775, Adjusted R-squared:  0.9765
## F-statistic:   957 on 7 and 154 DF,  p-value: < 2.2e-16
```

Regression equation:

$$\begin{aligned} Si = 98.81 &- (0.94 \cdot Na) - (0.99 \cdot Mg) - (1.00 \cdot Al) \\ &- (0.98 \cdot K) - (0.98 \cdot Ca) - (0.96 \cdot Ba) - (0.35 \cdot Fe) \end{aligned}$$

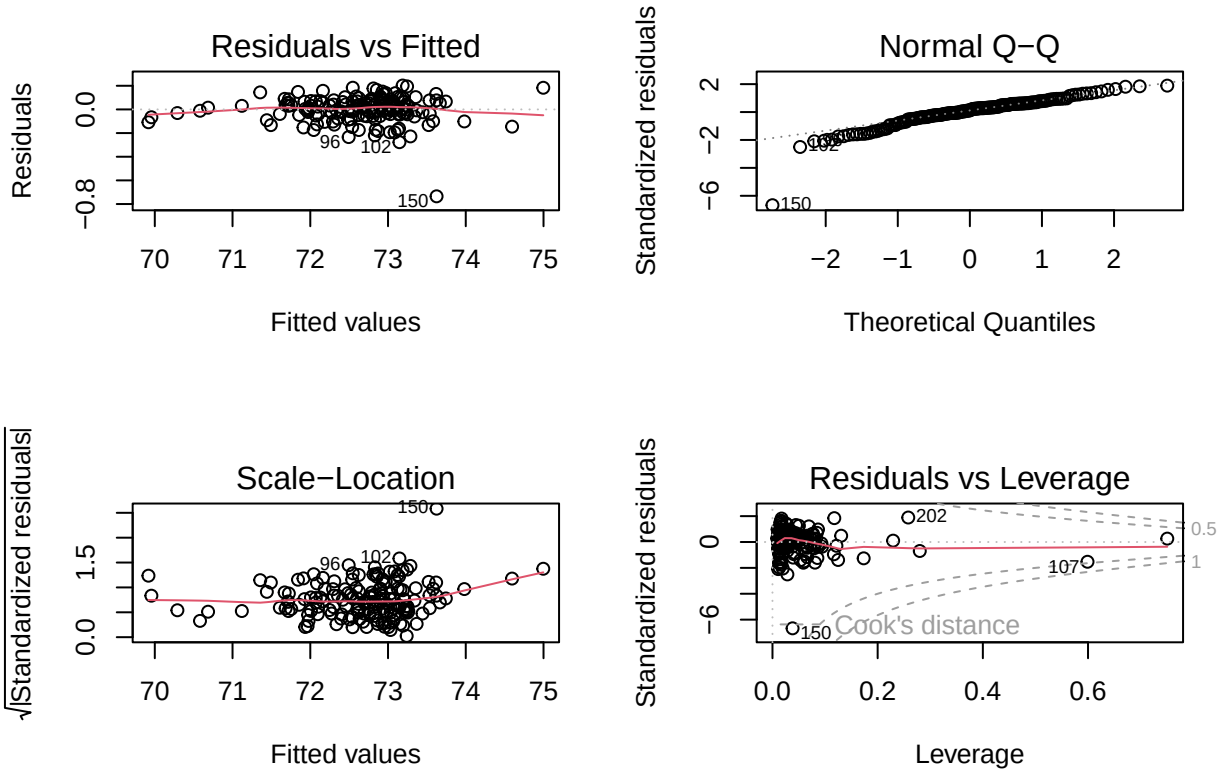
- (iii) Slopes and intercept interpretation

- determine the significant predictors at 10% level of significance

All of the variables in our best model are significant predictors of Si content at 10% confidence. This regression equation suggests that when none of the critical elements are present in the glass, 98.81% of the material will be Si The negative slopes in the equation means that an increase of the presence of any other element by one percent will decrease the Si content of the glass by roughly one percent, which makes sense. However, the exception is Fe, which only decreases Si content by 0.35% for every one percent change.

- (iv) Residual analysis
  - the four tests
  - four plots group together

```
par(mfrow = c(2,2))
plot(glass_model3)
```




```
# Check for Linearity of glass_model3 using rainbow test
raintest(glass_model3)
```

```
##
##  Rainbow test
##
## data:  glass_model3
## Rain = 0.59126, df1 = 81, df2 = 73, p-value = 0.9892
```

```r
# Check for normality of glass_model3 using shapiro test on residuals
# (This predicts residuals on training data)
real_resid <- residuals(glass_model3)
shapiro.test(real_resid) # violated
```

```
##
##  Shapiro-Wilk normality test
##
## data:  real_resid
## W = 0.8722, p-value = 1.541e-10
```

```r
# (This predicts residuals on TESTING data)
resids <- test.data$Si - predictions # This is what we want!
shapiro.test(resids)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  resids
## W = 0.97808, p-value = 0.448
```

```r
# Check for multicollinearity
vif(glass_model3)
```

```
##       Na       Mg       Al        K       Ca       Ba       Fe
## 1.857722 4.492997 2.718487 1.739089 3.400448 1.798373 1.071352
```

```r
dwtest(glass_model3)
```

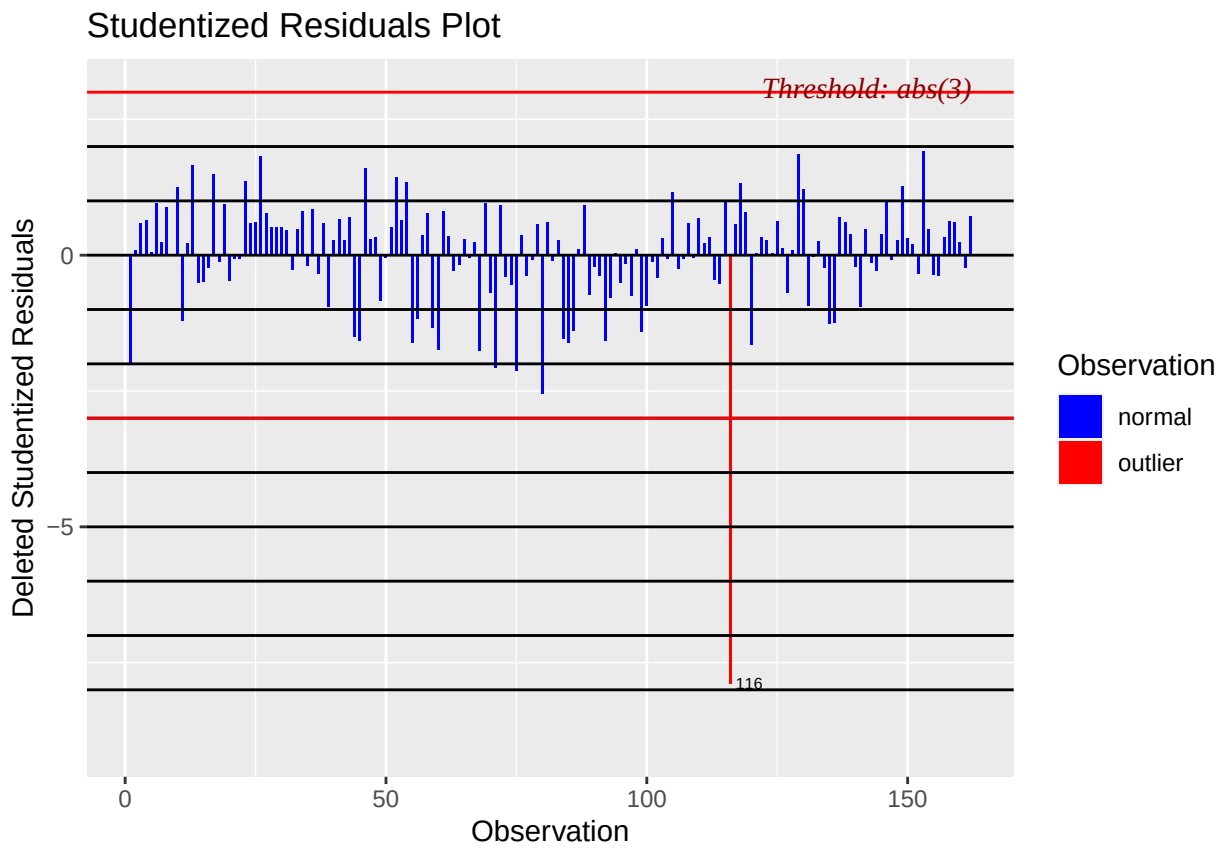
```
##
##  Durbin-Watson test
##
## data:  glass_model3
## DW = 2.0254, p-value = 0.501
## alternative hypothesis: true autocorrelation is greater than 0
```

```r
ols_test_breusch_pagan(glass_model3) # violated
```

```
##
##  Breusch Pagan Test for Heteroskedasticity
##  -----------------------------------------
##  Ho: the variance is constant
##  Ha: the variance is not constant
##
##              Data
##  -----------------------------
##  Response : Si
##  Variables: fitted values of Si
##
##          Test Summary
```

```
##  ------------------------------
##  DF            =     1
##  Chi2          =     22.46384
##  Prob > Chi2   =     2.14137e-06
```

```
ols_plot_resid_stud(glass_model3) # OUTLIER!!!!
```

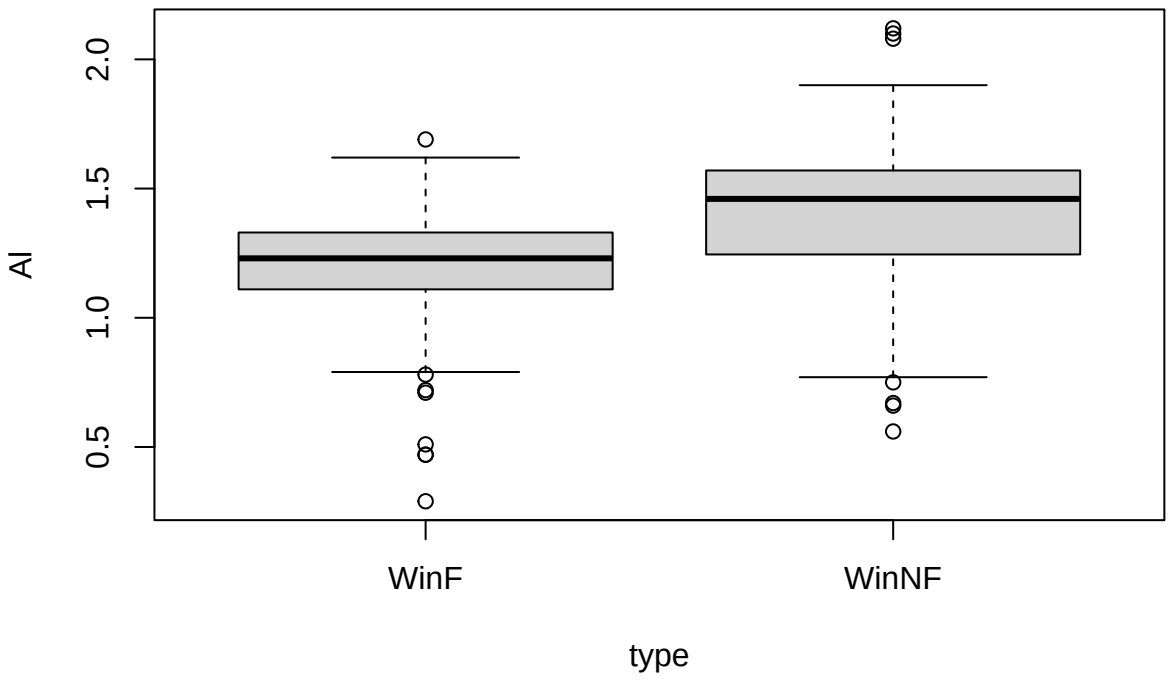


Based on the rainbow test on our model, the linearity assumption is not violated, as the null hypothesis of a linear fit is not rejected ($p > 0.05$). However, the results of a Shapiro test on the residuals of the model conclude they are not normal, as $p < 0.05$ here. Heteroscedasticity is also present in the residuals, as is an outlier. Thus, the assumptions on residuals of regression models to be normal do not hold.

Part (d)

```
glass_mutate <- glass %>%
  filter(type == "WinF" | type == "WinNF")

# Check 1 Independent of one another
boxplot(Al ~ type,
        data = glass_mutate,
        main = "Comparison of AL and types: WinF and WinNF")
```




```
# Check 2: Normality
shapiro.test(glass_mutate$Al)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  glass_mutate$Al
## W = 0.97374, p-value = 0.006643
```

```
# Check 3: Levene's test, The two groups have equal variance
var.test(Al ~ type,
         data = glass_mutate,
         conf.level = 0.9)
```

```
##
##  F test to compare two variances
##
## data:  Al by type
## F = 0.73628, num df = 69, denom df = 75, p-value = 0.1989
## alternative hypothesis: true ratio of variances is not equal to 1
## 90 percent confidence interval:
##  0.4991165 1.0901490
## sample estimates:
## ratio of variances
##          0.7362831
```

```r
# Perform a non-parametric test: Since the distribution of Al is not normal, use a
# non-parametric test, such as the Wilcoxon-Mann-Whitney test, to compare the
# medians of the two groups
wilcox_result <- wilcox.test(Al ~ type,
                             data = glass_mutate,
                             var.equal = TRUE,
                             conf.level = 0.9)
wilcox_result
```

```
##
##  Wilcoxon rank sum test with continuity correction
##
## data:  Al by type
## W = 1399, p-value = 7.875e-07
## alternative hypothesis: true location shift is not equal to 0
```

```r
p_value <- wilcox_result$p.value
```

```
##
## The distributions of the two groups are significantly different at the 10% significance level.
```

The Shaprio test reveals a deviation from normal distribution in Al, as evidenced by a p-value below 0.05. Consequently, a t-test comparing the means of the two groups cannot be conducted due to the non-normality. However, the two groups are independent and exhibit variances that are not significantly dissimilar, as indicated by an F test p-value exceeding 0.05. Thus, it is appropriate to employ a non-parametric test like the Wilcoxon-Mann-Whitney test to compare the medians of the two groups. Employing a 10% level of significance, the distributions of the two groups, WinF and WinNF, are markedly distinct since the p-value for the Wilcoxon test is significantly below 0.05.

Part (e):

```r
set.seed(123)
# Need 214 values
N <- 214

Na <- rnorm(n=N, mean = mean(glass$Na), sd(glass$Na))
Mg <- rnorm(n=N, mean = mean(glass$Mg), sd(glass$Mg))
Al <- rnorm(n=N, mean = mean(glass$Al), sd(glass$Al))
K <- rnorm(n=N, mean = mean(glass$K), sd(glass$K))
Ca <- rnorm(n=N, mean = mean(glass$Ca), sd(glass$Ca))
Ba <- rnorm(n=N, mean = mean(glass$Ba), sd(glass$Ba))
Fe <- rnorm(n=N, mean = mean(glass$Fe), sd(glass$Fe))

e <- rnorm(n=N, 0, 1)

# Saves coefficients from our model3 as variables
coef <- summary(glass_model3)$coefficients

y <- e + coef[1] + coef[2]*Na + coef[3]*Mg + coef[4]*Al + coef[5]*K +
  coef[6]*Ca + coef[7]*Ba + coef[8]*Fe
```

```
summary(y)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   65.84   71.04   72.44   72.56   74.43   80.36
```

```
# Simulated model
sim_model <- lm(y ~ Na + Mg + Al + K + Ca + Ba + Fe)
summary(sim_model)
```

```
##
## Call:
## lm(formula = y ~ Na + Mg + Al + K + Ca + Ba + Fe)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.4652 -0.6495 -0.0993  0.7263  3.1087
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 98.26225    1.37527  71.449  < 2e-16 ***
## Na          -0.89669    0.09343  -9.597  < 2e-16 ***
## Mg          -0.98889    0.04968 -19.905  < 2e-16 ***
## Al          -1.00191    0.14444  -6.936 5.10e-11 ***
## K           -1.12022    0.10732 -10.438  < 2e-16 ***
## Ca          -0.97894    0.04829 -20.271  < 2e-16 ***
## Ba          -0.79267    0.15098  -5.250 3.77e-07 ***
## Fe           1.04393    0.73822   1.414    0.159
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.035 on 206 degrees of freedom
## Multiple R-squared:  0.8386, Adjusted R-squared:  0.8331
## F-statistic: 152.9 on 7 and 206 DF,  p-value: < 2.2e-16
```

Simulated Model Regression equation:

$$\begin{aligned} Si = 98.32 - (0.89 \cdot Na) - (0.989 \cdot Mg) - (0.993 \cdot Al) \\ - (1.26 \cdot K) - (0.98 \cdot Ca) - (0.805 \cdot Ba) + (1.05 \cdot Fe) \end{aligned}$$

This regression equation suggests that when none of the critical elements are present in the glass, 98.32% of the material will be Si. The negative slopes in the equation means that an increase of the presence of any other element by one percent will decrease the Si content of the glass by roughly one percent, which makes sense.

It is clearly to see that our simulated regression has completely different coefficients for iron where for every one percent of increase in iron, the percentage of materials that are silicon in the glass will increases by 1.05%. (Certain chemical properties?)

```
# Check for Linearity of glass_model3 using rainbow test
raintest(sim_model)
```

```
##
##  Rainbow test
##
## data:  sim_model
## Rain = 0.81722, df1 = 107, df2 = 99, p-value = 0.8471
```

```
# Check for normality of glass_model3 using shapiro test on residuals
sim_resid <- residuals(sim_model)

shapiro.test(sim_resid) # violated
```

```
##
##  Shapiro-Wilk normality test
##
## data:  sim_resid
## W = 0.99524, p-value = 0.7433
```

```
# Check for multicollinearity
vif(sim_model)
```

```
##       Na       Mg       Al        K       Ca       Ba       Fe
## 1.031273 1.009741 1.010203 1.006343 1.023204 1.036901 1.028267
```

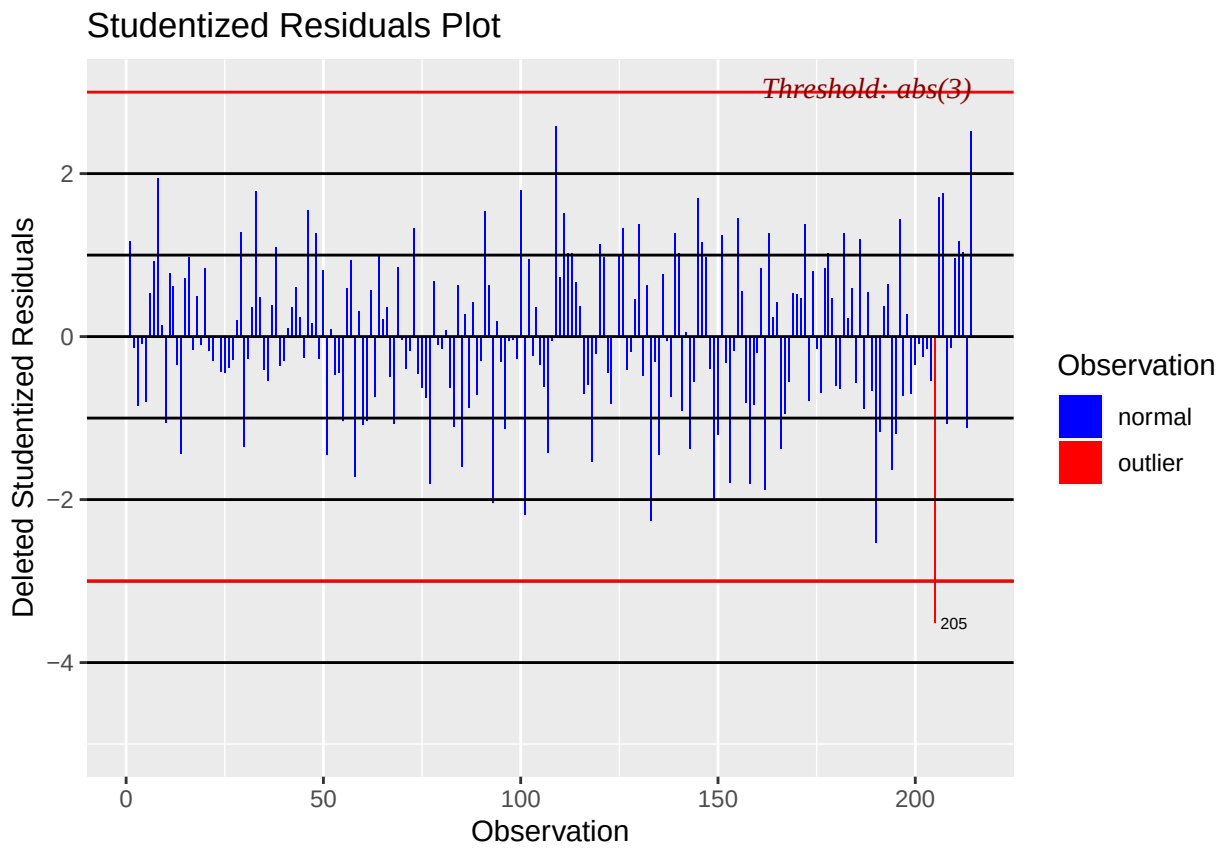
```
dwtest(sim_model)
```

```
##
##  Durbin-Watson test
##
## data:  sim_model
## DW = 1.9962, p-value = 0.4919
## alternative hypothesis: true autocorrelation is greater than 0
```

```
ols_test_breusch_pagan(sim_model) # violated
```

```
##
##  Breusch Pagan Test for Heteroskedasticity
##  -----------------------------------------
##  Ho: the variance is constant
##  Ha: the variance is not constant
##
##              Data
##  -----------------------------
##  Response : y
##  Variables: fitted values of y
##
##         Test Summary
##  ----------------------------
##  DF            =    1
##  Chi2          =    0.8257767
##  Prob > Chi2   =    0.3634966
```

```
ols_plot_resid_stud(sim_model) # OUTLIER!!!!
```



Problem 3

Part (a)

We choose to use 10-cross validation. Why we choose it

10-fold Cross-Validation

```
# Convert COMMIT to binary < median = 0, otherwise = 1.

new_Handout1 <- Handout1 %>%
  mutate(COMMIT = case_when(
    (COMMIT < median(COMMIT)) ~ 0,
    TRUE ~ 1))

# Select only the continuous numerical variables
new_Handout1 <- new_Handout1 %>%
  select(COMMIT, AGE, SALARY, CLASSSIZE, RESOURCES, AUTONOMY, CLIMATE, SUPPORT)
```

```r
# Training and testing method
set.seed(123)
# Create training and testing data
index <- createDataPartition(new_Handout1$COMMIT,
                             p = 0.8,
                             list = FALSE,
                             times=1)


new_Handout1 <- as.data.frame(new_Handout1)

train  <- new_Handout1[index, ]
test <- new_Handout1[-index, ]

train$COMMIT[train$COMMIT==1] <- "yes"
train$COMMIT[train$COMMIT==0] <- "no"

test$COMMIT[test$COMMIT==1] <- "yes"
test$COMMIT[test$COMMIT==0] <- "no"


# Convert outcome variable to factor for each data frame
train$COMMIT <- as.factor(train$COMMIT)
test$COMMIT <- as.factor(test$COMMIT)

# 10-fold Cross-validation method
ctrlspecs <- trainControl(method="cv",
                          number=10,
                          savePredictions="all",
                          classProbs=TRUE)

set.seed(1234)
cvmodel <- train(COMMIT ~ .,
                 data=train,
                 method="glm",
                 family = "binomial",
                 trControl=ctrlspecs)


# results1 <- data.frame( R2 = R2(predictions, test$COMMIT),
#             RMSE = RMSE(predictions, test$COMMIT),
#             MAE = MAE(predictions, test$COMMIT))
# # Results for 10-fold Cross-validation method
# results1

# Model for 10-fold Cross-validation method
summary(cvmodel)
```

```
##
## Call:
## NULL
##
## Deviance Residuals:
```

```
##      Min        1Q    Median        3Q       Max
## -2.06170  -0.90585   0.08378   0.93637   1.94307
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) -10.38779    5.07406  -2.047   0.0406 *
## AGE           0.14086    0.09033   1.559   0.1189
## SALARY        0.11065    0.09566   1.157   0.2474
## CLASSSIZE    -0.14105    0.06897  -2.045   0.0408 *
## RESOURCES     0.18733    0.13923   1.345   0.1785
## AUTONOMY      0.30463    0.12547   2.428   0.0152 *
## CLIMATE       0.21419    0.09829   2.179   0.0293 *
## SUPPORT      -0.19250    0.13418  -1.435   0.1514
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 166.36  on 119  degrees of freedom
## Residual deviance: 130.47  on 112  degrees of freedom
## AIC: 146.47
##
## Number of Fisher Scoring iterations: 4
```

```
print(cvmodel)
```

```
## Generalized Linear Model
##
## 120 samples
##   7 predictor
##   2 classes: 'no', 'yes'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 108, 108, 108, 108, 108, 108, ...
## Resampling results:
##
##   Accuracy  Kappa
##   0.675     0.35
```

```
varImp(cvmodel)
```

```
## glm variable importance
##
##           Overall
## AUTONOMY   100.00
## CLIMATE     80.43
## CLASSSIZE   69.90
## AGE         31.67
## SUPPORT     21.86
## RESOURCES   14.85
## SALARY       0.00
```

Model Accuracy

```
# Predict outcome using model from training data based on testing data
predictions <- predict(cvmodel, newdata=test)

# Create confusion matrix to assess model fit/performance on test data
con_matx <- confusionMatrix(data=predictions, test$COMMIT)
con_matx
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction no yes
##        no  13   4
##        yes  2  11
##
##                Accuracy : 0.8
##                  95% CI : (0.6143, 0.9229)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : 0.0007155
##
##                   Kappa : 0.6
##
##  Mcnemar's Test P-Value : 0.6830914
##
##             Sensitivity : 0.8667
##             Specificity : 0.7333
##          Pos Pred Value : 0.7647
##          Neg Pred Value : 0.8462
##              Prevalence : 0.5000
##          Detection Rate : 0.4333
##    Detection Prevalence : 0.5667
##       Balanced Accuracy : 0.8000
##
##        'Positive' Class : no
##
```

Part (b)

Our objective is to optimize the cost function in order to identify the most economical approach for fulfilling the Christmas order. This entails striving for the lowest cost per day to achieve maximum cost efficiency. By minimizing expenses at each factory, we can ensure a cost-effective process for meeting the demands of the Christmas order.

```
factories <- c("Factory A", "Factory B", "Factory C")
toys <- c("Cars", "Animals", "Robots")

# Define the coefficients of the objective function
costs <- c(1000, 2100, 1500)

# Define the constraint matrix
constraint_matrix <- matrix(c(30, 20, 30,
```

```
                              40, 50, 10,
                              50, 40, 15), nrow = 3, byrow = TRUE)

# Define the right-hand side of the constraints
constraint_limits <- c(5000, 3000, 2500)

# Set the direction of optimization (minimization)
constraint_directions <- c(">=", ">=", ">=")
direction <- "min"

# Solve the linear programming problem
min_solution <- lp(direction = direction,
                objective.in = costs,
                const.mat = constraint_matrix,
                const.dir = constraint_directions,
                const.rhs = constraint_limits,
                compute.sens=TRUE)


# Check if a solution was found
if (min_solution$status == 0) {
  # Print the optimal solution
  cat("The optimal solution is:\n")
  cat("Factory A:",min_solution$solution[1], "days","\n")
  cat("Factory B:",min_solution$solution[2], "days","\n")
  cat("Factory C:",min_solution$solution[3], "days","\n\n")
  # Print the minimum cost
  cat("The value of the objective function at the optimal solution is:\n")
  cat("Minimum cost: $",min_solution$objval)
} else {
  # No feasible solution found
  print("No feasible solution found.")
}
```

```
## The optimal solution is:
## Factory A: 166.6667 days
## Factory B: 0 days
## Factory C: 0 days
##
## The value of the objective function at the optimal solution is:
## Minimum cost: $ 166666.7
```

The most efficient approach entails running Factory A for approximately 166.67 days, resulting in a minimal cost of $166,666.70, whereas Factory B and Factory C remain non-operational. By adopting this strategy, the company can achieve the most cost-effective outcome.

When considering the optimal solution, it is important to analyze the cost implications of running the factories for different durations. In this scenario, operating Factory for approximately 166.67 days leads to the minimum overall cost. By halting the operations of Factory B and Factory C, the company can avoid additional expenses associated with their functioning.

```
# Create an empty data frame to store the results
result_df <- data.frame()
```

```r
factories <- c("Factory A", "Factory B", "Factory C")
toys <- c("Cars", "Animals", "Robots")



# Iterate over the range of 1:3
for (i in 1:3) {

  # Store values for rows and columns in the data frame
  row_df <- data.frame(Factory = factories[i],
                       min_days = min_solution$solution[i],
                       min_cost = min_solution$solution[i]*costs[i])


  result_df <- rbind(result_df, row_df)

}

colnames(result_df) <- c("",
                         "Minimum # of days",
                         "Minimum costs")

# Print the resulting data frame
cat("The optimal solution is:")
```

```
## The optimal solution is:
```

```r
print(result_df)
```

```
##             Minimum # of days Minimum costs
## 1 Factory A          166.6667      166666.7
## 2 Factory B            0.0000           0.0
## 3 Factory C            0.0000           0.0
```

```r
# Sensitivity Analysis

min_solution$sens.coef.from
```

```
## [1]    0.0000  666.6667 1000.0000
```

```r
min_solution$sens.coef.to
```

```
## [1] 1.5e+03 1.0e+30 1.0e+30
```

Part (c)

```r
compute_weibull_stats <- function(shape, scale) {
  # Compute the mean
  mean_value <- scale * gamma(1 + (1/shape))
```

```r
  # # Compute the median
  # median_value <- scale * qweibull(0.5, shape, scale)


  # Compute the median
  median_value <- scale * (log(2)^(1/shape))


  # Compute the mode
  if (shape > 1)
    mode_value <- scale * ((shape - 1) / shape)^(1/shape)
  else
    mode_value <- 0

  # Compute the variance
  variance_value <- scale^2 * (gamma(1 + (2/shape)) - (gamma(1 + (1/shape)))^2)

  # Return the computed statistics as a named list
  return(list(mean = mean_value,
              median = median_value,
              mode = mode_value,
              variance = variance_value))
}

# lambda = scale
# K = shape

# Example usage
scale_param <- 6
shape_param <- 1

stats <- compute_weibull_stats(shape_param, scale_param)

# Accessing the computed statistics
mean_value <- stats$mean
median_value <- stats$median
mode_value <- stats$mode
variance_value <- stats$variance

# Printing the computed statistics
cat("Mean:", mean_value, "\n")
```

```
## Mean: 6
```

```r
cat("Median:", median_value, "\n")
```

```
## Median: 4.158883
```

```r
cat("Mode:", mode_value, "\n")
```

```
## Mode: 0
```

```r
cat("Variance:", variance_value, "\n")
```

```
## Variance: 36
```

Maximum likelihood estimator (MLE) of the parameters from the Weibull distribution

Probability Density Function ($f(x;\lambda,k)$) for the Weibull distribution:

$$f(x;\lambda,k) = \begin{cases} \frac{k}{\lambda}\left(\frac{x}{\lambda}\right)^{k-1} e^{-\left(\frac{x}{\lambda}\right)^k}, & x \geq 0 \\ 0, & x < 0 \end{cases}$$

The likelihood function ($L_{\hat{x}}(\lambda,k)$):

$$\begin{aligned} L_{\hat{x}}(\lambda,k) &= \prod_{i=1}^{n} \frac{k}{\lambda}\left(\frac{x_i}{\lambda}\right)^{k-1} e^{-\left(\frac{x_i}{\lambda}\right)^k} \\ &= \left(\frac{k}{\lambda}\right)^n \left(\frac{1}{\lambda^{k-1}}\right)^n \left(\prod_{i=1}^{n} x_i^{k-1}\right) \left(e^{-\sum_{i=1}^{n}\left(\frac{x_i}{\lambda}\right)^k}\right) \\ &= \frac{k^n}{\lambda^{nk}} e^{-\sum_{i=1}^{n}\left(\frac{x_i}{\lambda}\right)^k} \prod_{i=1}^{n} x_i^{k-1} \end{aligned}$$

The log-likelihood function ($\ln L_{\hat{x}}(\lambda,k)$):

$$\ln L_{\hat{x}}(\lambda,k) = n\ln(k) - nk\ln(\lambda) - \sum_{i=1}^{n}\left(\frac{x_i}{\lambda}\right)^k + (k-1)\sum_{i=1}^{n}\ln x_i$$

Partial derivative of the log-likelihood function with respect to λ:

$$\frac{\partial \ln L_{\hat{x}}(\lambda,k)}{\partial\lambda} = -\frac{nk}{\lambda} + k\sum_{i=1}^{n}\frac{x_i^k}{\lambda^{k+1}}$$

Solving for the desired parameter of λ by setting the partial derivative equal to zero:

$$\begin{aligned}
\frac{\partial \ln L_{\hat{x}}(\lambda, k)}{\partial \lambda} &= 0 \\
-\frac{nk}{\lambda} + k\sum_{i=1}^{n} \frac{x_i^k}{\lambda^{k+1}} &= 0 \\
-\frac{nk}{\lambda} + \frac{k}{\lambda}\sum_{i=1}^{n} \left(\frac{x_i}{\lambda}\right)^k &= 0 \\
-n + \sum_{i=1}^{n} \frac{x_i^k}{\lambda^k} &= 0 \\
\frac{1}{\lambda^k}\sum_{i=1}^{n} x_i^k &= n \\
\frac{1}{n}\sum_{i=1}^{n} x_i^k &= \lambda^k
\end{aligned}$$

Therefore the estimator $\hat{\lambda}$ is:

$$\hat{\lambda} = \left(\frac{1}{n}\sum_{i=1}^{n} x_i^k\right)^{\frac{1}{k}}$$

Plugging in $\hat{\lambda}$ into the log-likelihood function ($\ln L_{\hat{x}}(\lambda, k)$) and then differentiating with respect to k in order to find the estimator $\hat{k}$:

$$\begin{aligned}
\frac{\partial \ln L_{\hat{x}}(\lambda, k)}{\partial k} &= \frac{\partial}{\partial k}\left[n\ln k - nk\ln\lambda - \sum_{i=1}^{n}\left(\frac{x_i}{\lambda}\right)^k + (k-1)\sum_{i=1}^{n}\ln x_i\right] \\
&= \frac{\partial}{\partial k}\left[n\ln k - nk\ln\left[\left(\frac{1}{n}\sum_{i=1}^{n} x_i^k\right)^{\frac{1}{k}}\right] - \frac{\sum_{i=1}^{n} x_i^k}{\left[\left(\frac{1}{n}\sum_{i=1}^{n} x_i^k\right)^{\frac{1}{k}}\right]^k} + (k-1)\sum_{i=1}^{n}\ln x_i\right] \\
&= \frac{\partial}{\partial k}\left[n\ln k - \frac{nk}{k}\ln\left(\frac{1}{n}\sum_{i=1}^{n} x_i^k\right) - \frac{\sum_{i=1}^{n} x_i^k}{\left(\frac{1}{n}\sum_{i=1}^{n} x_i^k\right)} + (k-1)\sum_{i=1}^{n}\ln x_i\right] \\
&= \frac{\partial}{\partial k}\left[n\ln k - n\ln\left(\frac{1}{n}\sum_{i=1}^{n} x_i^k\right) - n + (k-1)\sum_{i=1}^{n}\ln x_i\right] \\
&= \frac{n}{k} - \left(\frac{n\sum_{i=1}^{n} x_i^k \ln x_i}{\sum_{i=1}^{n} x_i^k}\right) + \sum_{i=1}^{n}\ln x_i \\
&= \frac{1}{k} - \left(\frac{\sum_{i=1}^{n} x_i^k \ln x_i}{\sum_{i=1}^{n} x_i^k}\right) + \frac{1}{n}\sum_{i=1}^{n}\ln x_i
\end{aligned}$$

Solving for the desired parameter of k by setting the partial derivative equal to zero:

$$\frac{\partial \ln L_{\hat{x}}(\lambda, k)}{\partial k} = 0$$

$$\frac{1}{k} - \left(\frac{\sum_{i=1}^{n} x_i^k \ln x_i}{\sum_{i=1}^{n} x_i^k} \right) + \frac{1}{n} \sum_{i=1}^{n} \ln x_i = 0$$

$$\left(\frac{\sum_{i=1}^{n} x_i^k \ln x_i}{\sum_{i=1}^{n} x_i^k} \right) - \frac{1}{n} \sum_{i=1}^{n} \ln x_i = \frac{1}{k}$$

Therefore the estimator $\hat{k}$ is:

$$\hat{k} = \left[\frac{\sum_{i=1}^{n} x_i^k \ln x_i}{\sum_{i=1}^{n} x_i^k} - \frac{1}{n} \sum_{i=1}^{n} \ln x_i \right]^{-1}$$